CSCI 340 HW 2
Spring 2023

Collaboration Policy: Individual Assignment
Total points: 100 Points

1 HTTP Server Specification

Using the C server template code attached to the Dropbox, modify the template code to become a simple HTTP server that responds to simple POST and GET requests sent from a standard web browser (eg. Chrome, Safari, Firefox).

Your HTTP server **must**:

- process HTTP GET and POST messages (request and response)
- use the *fork()* system call to spawn a new child process for each client (ie. web browser)
- implement a child signal handler that "reaps" the child (ie. remove the child process from the process table) when it has completed. HINT: look at *wait()* and *waitpid()*.
- include support for loading basic HTML webpages using a GET message

Your HTTP server does **not need to**:

- include support for cookies
- include support for conditional GET
- include support for multipart request operations
- include support for encryption/decryption capabilities
- include support for media, (eg. images, movies, etc.)

You **must use and modify** the C code provided to you, and it must run on a Ubuntu operating system. You **may use** any standard library function supported in the C language specification. You **cannot change** the Makefile (eg. change the compiler from gcc to g++).

For most of you, this will likely be the second C program you have written. **Give yourself plenty of time**. **Do not copy paste existing code found on the web**. Your solution must utilize the provided code and framework. Rewriting everything provided to you and/or pasting solutions from the web will likely lead to a zero for the assignment. This is not a "search the web" exercise. I may ask you as part of the assessment of your solution to explain portions of your submitted solution.

2 HTTP GET

From a web browser, if you enter the following URL

`http://<server>:<port>/?first=anthony&last=leclerc`

with <server> and <port> replaced with the IP address and port number of your running HTTP server, then your server should return the following HTML shown in Appendix A.

In general, your HTTP server **must** parse any number of key/value pairs appended to the URL

`http://<server>:<port>/?key=value&key=value& ...`

and then return a response message where the entity body outputs the key/value pairs in a HTML formatted table (See Appendix A for the previous example). For more information about the structure of the HTTP request message see Appendix D, or more information about the structure of the HTTP response message see Appendix E.

3 HTTP POST

From a web browser, if you enter the following URL

`http://<server>:<port>/SimplePost.html`

with <server> and <port> replaced with the IP address and port number of your running HTTP server, then your server should return a webpage that is similar to that seen in Appendix C

When you submit the webpage with an HTTP <form> tag that uses a HTTP POST operation, your HTTP server **must** parse any number of key/value pairs in the entity body of the request message, and then return a response message where the entity body outputs the key/value pairs in a HTML formatted table (See Appendix C for an example).

For more information about the structure of the HTTP request message see Appendix D, or more information about the structure of the HTTP response message see Appendix E.

4 Additional Information

For more information about the HTTP protocol:

- The Internet engineering task force (IETF) request for comment (RFC) specification (`https://tools.ietf.org/html/rfc7231`). Essentially, this is official document that fully describes the protocol and how to implement it.

To help troubleshoot connection and protocol implementation problems your HTTP server may encounter, I suggest using Wireshark (`https://www.wireshark.org`) to follow/analyze the TCP connections, and to inspect the packets sent between the client (ie. the web browser) and your HTTP server.

5 Submission

Create a tarball that includes all the code required to compile and run your HTTP server (this includes your Makefile). To do this, enter on the command line:

```
make tarball
```

Submit the tarball file (via OAKS) to the Dropbox setup for this assignment by the due date. You may resubmit the tarball file as many times as you like. I will only grade the most recent submission.

6 Grading Rubric

HTTP server (with *fork()*, parsing, and "reap") compiles	30 points
HTTP server runs without errors	10 points
HTTP GET test cases (3 cases 10 points each)	30 points
HTTP POST test cases (3 cases 10 points each)	30 points
	100 points

In particular, the assignment will be graded as follow, if the submitted solution

- Does not compile: 0 of 100 points
- Compiles but does not run: 30 of 100 points
- Compiles and runs with no errors: 40 of 100 points
- Passes all 6 test cases developed by instructor: 100 of 100 points.

A Example HTML returned by HTTP GET request

```
<html>
  <body>
    <h1>GET Operation</h1>
    <table cellpadding=5 cellspacing=5 border=1>
      <tr>
        <td><b>first</b></td>
        <td>anthony</td>
      </tr>
      <tr>
        <td><b>last</b></td>
        <td>leclerc</td>
      </tr>
      <table>
  </body>
</html>
```

B Example SimplePost.html file

```
<html>
  <head>
    <meta charset="UTF-8">
    <title>Simple Post</title>
  </head>
  <body>
    <form method="POST" action="<server>">
      <b>first</b> </b><input type="text" name="first"><br>
      <b>last</b> </b><input type="text" name="last"><br>
      <input type="submit">
    </form>
  </body>
</html>
```

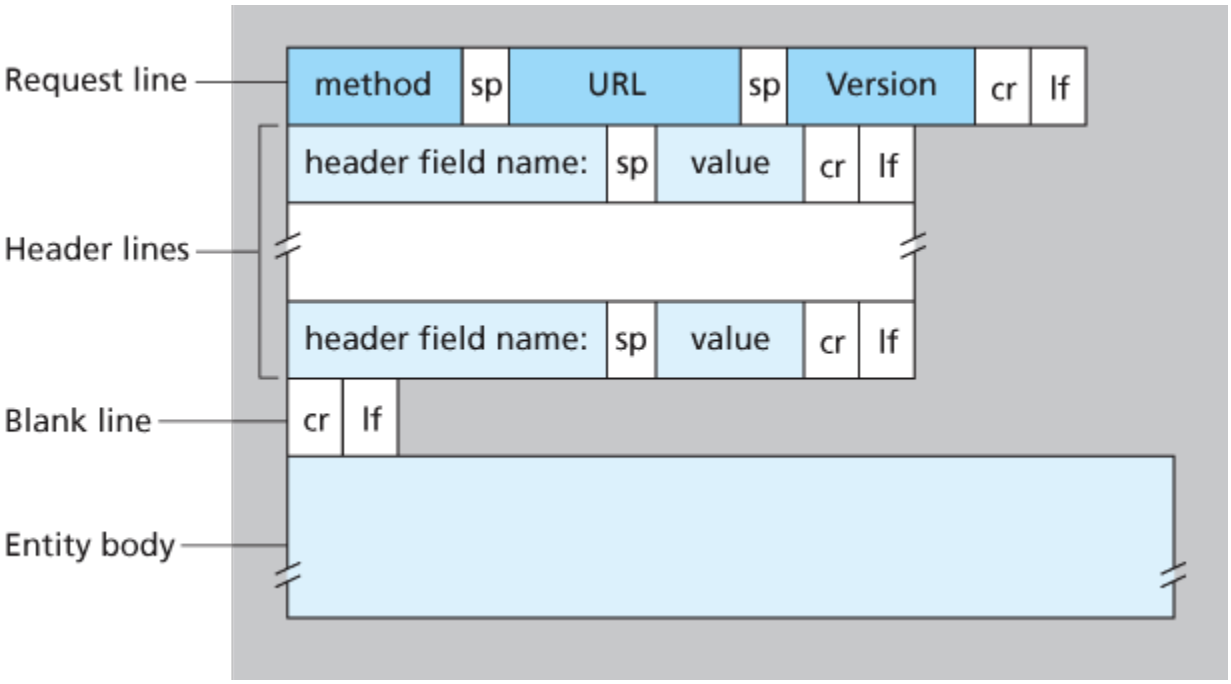
Where <server> is replaced with the IP address of the machine your HTTP server is running on.

C Example HMTL returned by HTTP POST request

```
<html>
  <body>
    <h1>POST Operation</h1>
    <table cellpadding=5 cellspacing=5 border=1>
      <tr>
        <td><b>first</b></td>
        <td>anthony</td>
      </tr>
      <tr>
        <td><b>last</b></td>
        <td>leclerc</td>
      </tr>
```

```
      <table>
  </body>
</html>
```

D HTTP Request Message Format



E HTTP Response Message Format